CSE 341 Section 4

Nicholas Shahan

Spring 2016

Adapted from slides by Cody A. Schroeder, and Dan Grossman

Today's Agenda

- Mutual Recursion
- Module System Example
  - Namespace Organization
  - Preserving Invariants
- Practice with Currying and High Order Functions

Mutual Recursion

- What if we need function f to call g, and function g to call f?
- This is a common idiom

```
fun earlier x =
      ...
      later x
      ...
fun later x =
      ...
      earlier x
      ...
```

Unfortunately this does not work ☹

Mutual Recursion Workaround

- We can use higher order functions to get this working
- It works, but there has got to be a better way!

```
fun earlier (f, x) =
      ...
      f x
      ...
fun later x =
      ...
      earlier (later, x)
      ...
```

Mutual Recursion with **`and`**

- SML has a keyword for that
- Works with mutually recursive **`datatype`** bindings too

```
fun earlier x =
      ...
      later x
      ...
and later x =
      ...
      earlier x
      ...
```

Module System

- Good for organizing code, and managing namespaces (useful, relevant)
- Good for maintaining invariants (interesting)

Currying and High Order Functions

- List.map!
- List.filter!
- List.foldl!
- Emacs unite!